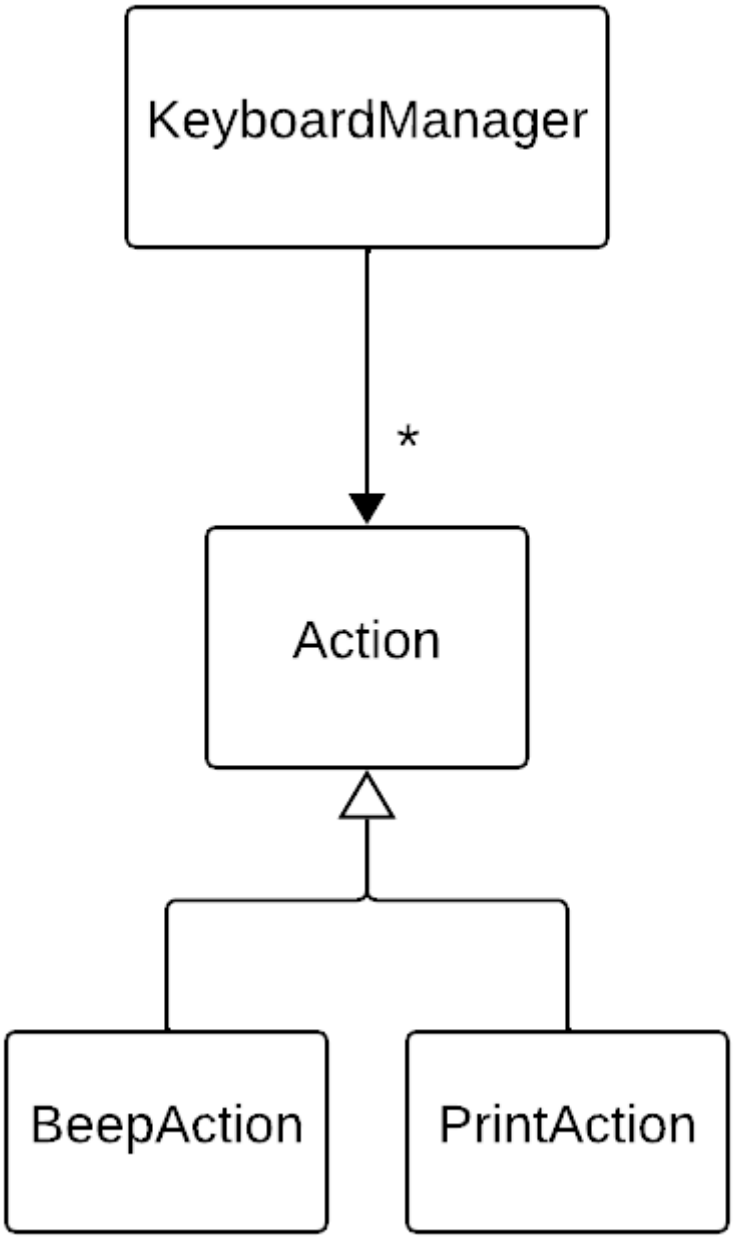
פתרון למבחן במת"מ – אביב 2024 מועד א'

שאלה 1

סעיף א:



סעיף ב:

```
class Action{
public:
    virtual ~Action() = default;
    virtual void applyAction() = 0;
};

class PrintAction : public Action{
private:
    string msg;
public:
    PrintAction(const string& message): msg(message) {}

    void applyAction() override{
        cout << msg << endl;
    }
};

class BeepAction : public Action{
public:
    void applyAction() override{
        beep();
    }
};
```

סעיף ג:

```
class KeyboardManager{
private:
    map<string, unique_ptr<Action>> keysMap;
public:
    void bind(const string& key, unique_ptr<Action> action){
        keysMap[key] = std::move(action);
    }

    void press(const string& key){
        if(keysMap.count(key) == 0) return;
        keysMap[key]->applyAction();
    }
};
```

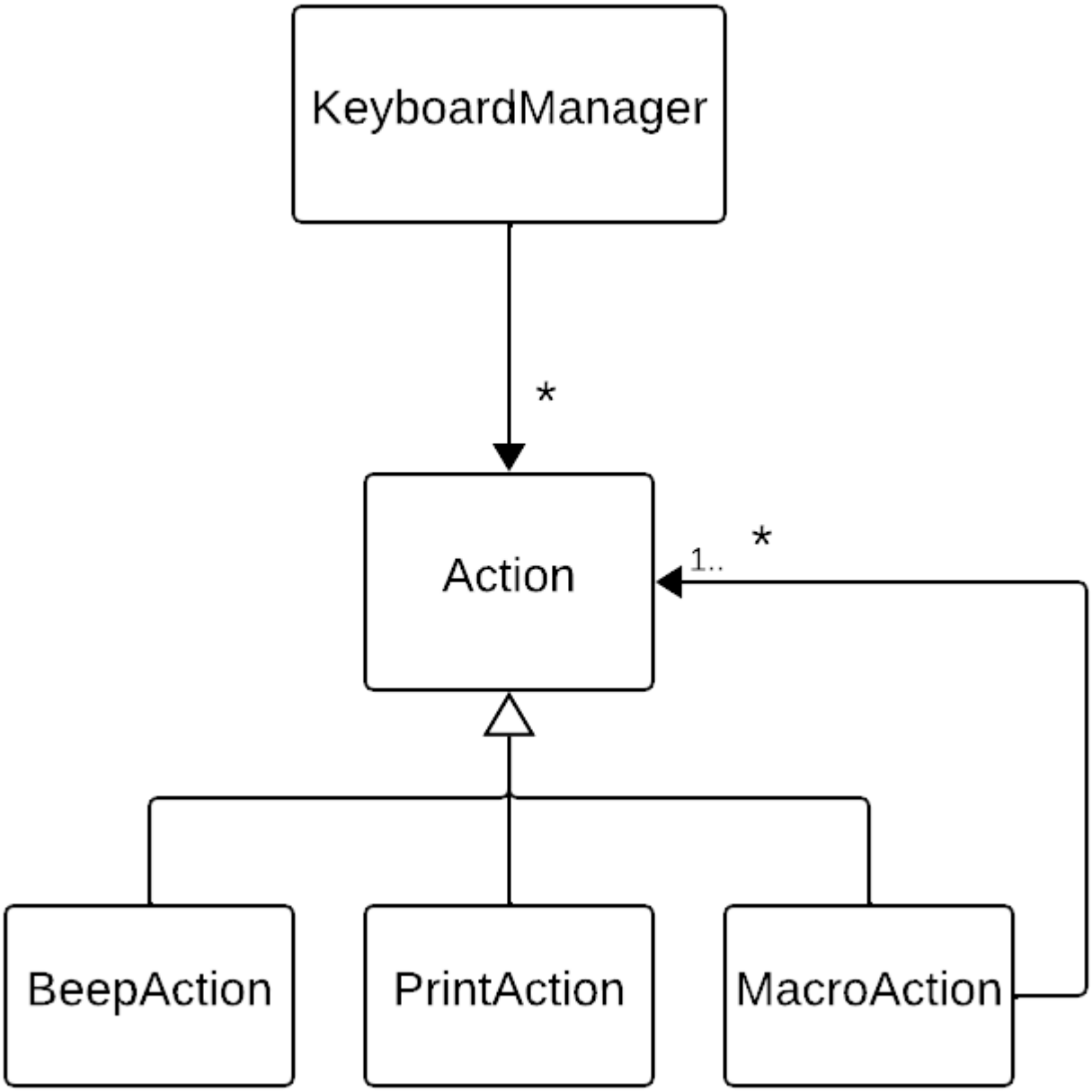
סעיף ד:

```
int main(){
    KeyboardManager manager;
    string key1 = "A";
    manager.bind(key1, make_unique<BeepAction>());
    string key2 = "B";
    manager.bind(key2, make_unique<PrintAction>("oopsy..."));
    manager.press(key1);
    manager.press(key2);
    return 0;
}
```

סעיף ה:

לא ניתן להעתיק אובייקטים מסוג KeyboardManager.
הגדרנו map שטיפוס ה-value הוא unique_ptr<Action>, ובמהלך ניסיון ההעתקה של ה-map תתבצע קריאה גם לבנאי ההעתקה של unique_ptr<T>, אך לא קיים בנאי העתקה מתאים כי למדנו שבנאי ההעתקה של unique_ptr<T> מסומן ב-"=delete".

סעיף ו:



סעיף ז:

```
class MacroAction : public Action{
private:
    vector<unique_ptr<Action>> actions;
public:
    MacroAction(vector<unique_ptr<Action>>& actions_arg){
        for (auto& action : actions_arg) {
            this->actions.push_back(std::move(action));
        }
    }

    void applyAction() override{
        for(auto& action : actions){
            action->applyAction();
        }
    }
};
```

שאלה 2

סעיף א:

```
template <typename T>
class MemManager {
private:
    T* data;
    int numBlock;
    int blockSize;
    T* follow;

public:
    MemManager(int numBlock, int blockSize){
        if(numBlock <= 0 || blockSize <= 0){
          throw invalid_argument("Invalid argument");
        }
        this->numBlock = numBlock;
        this->blockSize = blockSize;
        data = new T[numBlock * blockSize];
        follow = new T[numBlock];
        for(int i = 0; i < numBlock; i++){
            follow[i] = 0;
        }
    }

    ~MemManager(){
        delete[] data;
        delete[] follow;
    }
```

```
    T* acquire(){
        int t = -1;
        for(int i = 0; i < numBlock; i++){
            if(follow[i] == 0){
                t = i;
                break;
            }
        }
        if(t == -1) throw NoMoreBlocks();
        follow[t] = 1;
        return data + t * blockSize;
    }

    void release(const T* block){
        int i;
        for(i = 0; i < numBlock; i++){
            if(data + i * blockSize == block) break;
        }
        follow[i] = 0;
    }
};

class NoMoreBlocks : public exception {
public:
    const char* what() const throw(){
        return "No more blocks";
    }
};
```

סעיף ב:

לא ניתן להסתפק בבנאי העתקה ובאופרטור השמה הדיפולטיים.
במקרה של השמה או העתקה בין מופעים של MemManager, תתבצע השמה של מצביע לזיכרון דינאמי שאחד האובייקטים הקצה. ואז שינוי בזיכרון זה על ידי האובייקט המועתק או המושם ידרוס תאים בזיכרון בו השתמש האובייקט הראשון. בנוסף, אם רק אחד האובייקטים יהרס גם הזיכרון הדינאמי ישוחרר, ואז נשארנו עם אובייקט שמחזיק מצביע לזיכרון ששוחרר, וגישה אליו תגרום לבעיות זיכרון והתנהגות לא מוגדרת.

שאלה 3

סעיף א + ב:

```
class Student:
    def __init__(self, name, exam_grade, hw_grades):
        self.name = name
        self.exam_grade = exam_grade
        self.hw_grades = hw_grades.copy()

    def get_final_grade(self, exam_weight, hw_weight):
        examf = self.exam_grade * exam_weight
        hwf = sum([x*y for x,y in zip(self.hw_grades, hw_weight)])
        return examf + hwf
```

סעיף ג:

```
def print_students_in_dir(input_dir_path, exam_weight, hw_weight):
    fileList = os.listdir(input_dir_path)
    for file in fileList:
        if file.endswith(".json"):
           student = load_student_from_path(input_dir_path + os.sep + file)
             name = student.name + ":"
           print(name, student.get_final_grade(exam_weight, hw_weight))
\
```

סעיף ד:

```
def main(path):
    try:
        print_students_in_dir(path, exam_weight, hw_weight)
    except :
        print("An error has occurred", file=sys.stderr)


if __name__ == "__main__":
    main(sys.argv[1])
```